

PATENT ABSTRACTS OF JAPAN

(11)Publication number : 10-162013
 (43)Date of publication of application : 19.06.1998

(51)Int.Cl. G06F 17/30

(21)Application number : 08-317918
 (22)Date of filing : 28.11.1996

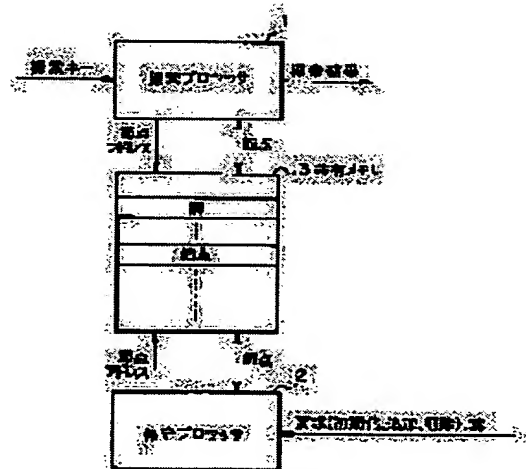
(71)Applicant : NIPPON TELEGR & TELEPH CORP <NTT>
 (72)Inventor : TAKAHASHI NAOHISA
 MARUYAMA MITSURU
 SANEI TAKESHI
 OGURA TAKESHI
 KAWANO TETSUO
 YAGI SATORU

(54) DIGITAL SEARCHING DEVICE

(57)Abstract:

PROBLEM TO BE SOLVED: To provide a digital searching device which executes a searching processing on a digital search tree at a high speed even if the number of headers in the digital search tree is increased, even if the request frequency of the searching processing, an elimination processing and an addition processing increase or even if the requests of the searching processing are continuously outputted.

SOLUTION: In the digital searching device, the digital search tree is kept in a common memory 3 and a maintenance processor 2 executes the initialization processing of the digital search tree, the addition processing of leaves and nodes and the elimination processing of the leaves and the nodes. A searching processor 1 executes the searching processing of the digital search tree. The maintenance processor 2 and the searching processor 1 independently operate and they access to the common memory 3 so as to operate the digital search tree in parallel.



LEGAL STATUS

[Date of request for examination] 21.12.1998
 [Date of sending the examiner's decision of rejection] 21.11.2001
 [Kind of final disposal of application other than the examiner's decision of rejection or application converted registration]
 [Date of final disposal for application]
 [Patent number] 3284064
 [Date of registration] 01.03.2002
 [Number of appeal against examiner's decision of rejection] 2001-22980
 [Date of requesting appeal against examiner's decision of rejection] 21.12.2001
 [Date of extinction of right]

BEST AVAILABLE COPY

Copyright (C); 1998,2003 Japan Patent Office

(19) 日本国特許庁 (J P)

(12) 公開特許公報 (A)

(11) 特許出願公開番号

特開平11-73329

(43) 公開日 平成11年(1999) 3月16日

(51) Int.Cl.⁸

G 0 6 F 9/45
12/08

識別記号

F I

G 0 6 F 9/44
12/08

3 2 2 H
S

審査請求 未請求 請求項の数 6 O L (全 23 頁)

(21) 出願番号 特願平10-13878

(22) 出願日 平成10年(1998) 1月27日

(31) 優先権主張番号 特願平9-166777

(32) 優先日 平 9 (1997) 6月24日

(33) 優先権主張国 日本 (J P)

(71) 出願人 000005821

松下電器産業株式会社

大阪府門真市大字門真1006番地

(72) 発明者 津幡 真太郎

大阪府門真市大字門真1006番地 松下電器
産業株式会社内

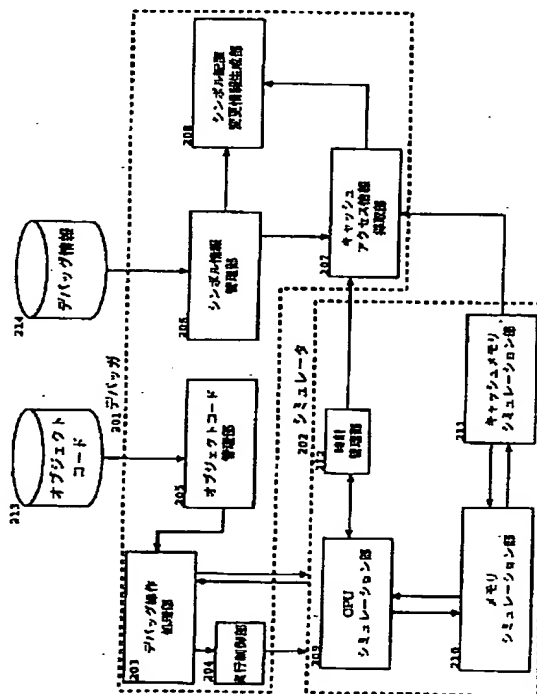
(74) 代理人 弁理士 宮井 暎夫

(54) 【発明の名称】 ソフトウェア開発支援システム

(57) 【要約】

【課題】 キャッシュメモリへのアクセスの状況を高級言語レベルで検出し、キャッシュメモリヒット率が向上するようにシンボルの割付情報を生成することにより、キャッシュメモリ搭載プロセッサ上のプログラム開発を支援するソフトウェア開発支援システムを提供する。

【解決手段】 プロセッサの動作をシミュレーションするシミュレータ202と、キャッシュメモリへのアクセスに関する情報を採取するキャッシュアクセス情報採取部207と、プログラムのシンボル情報を管理するシンボル情報管理部206と、プログラムのシンボルの配置を変更するためのシンボル配置変更情報を生成するシンボル配置変更情報生成部208とを備えている。



1

【特許請求の範囲】

【請求項1】 キャッシュメモリを搭載するプロセッサ上で実行されるプログラムのソフトウェア開発支援システムであって、

前記プロセッサの動作をシミュレーションするシミュレーション手段と、このシミュレーション手段から前記キャッシュメモリへのアクセスに関する情報を採取するキャッシュアクセス情報採取手段と、前記プログラムのシンボル情報を管理するシンボル情報管理手段と、前記プログラムのシンボルの配置を変更するためのシンボル配置変更情報を生成するシンボル配置変更情報生成手段とを備えたソフトウェア開発支援システム。

【請求項2】 キャッシュアクセス情報採取手段は、キャッシュメモリでのエントリの競合性ミスに関する情報を採取する請求項1記載のソフトウェア開発支援システム。

【請求項3】 シンボル配置変更情報生成手段は、キャッシュメモリでのエントリの競合性ミスを低減するための情報を生成する請求項1または請求項2記載のソフトウェア開発支援システム。

【請求項4】 キャッシュメモリを搭載するプロセッサ上で実行されるプログラムのソフトウェア開発支援システムであって、

前記プロセッサの動作をシミュレーションするシミュレーション手段と、このシミュレーション手段から前記キャッシュメモリへのアクセスに関する情報を採取するキャッシュアクセス情報採取手段と、前記プログラムのシンボル情報を管理するシンボル情報管理手段と、前記シンボル情報を用いて前記キャッシュアクセスの情報を表示する表示手段とを備えたソフトウェア開発支援システム。

【請求項5】 キャッシュアクセス情報採取手段は、アクセス対象のアドレスおよびアクセス時のプログラムカウンタ値、スタックポインタ値ならびにアクセスがヒットかどうかの情報を採取する請求項4記載のソフトウェア開発支援システム。

【請求項6】 表示手段は、キャッシュメモリでのエントリ別に分類して表示する請求項4または請求項5記載のソフトウェア開発支援システム。

【発明の詳細な説明】

【0001】

【発明の属する技術分野】 この発明は、キャッシュメモリを搭載するプロセッサ上で実行されるプログラムの開発を支援するソフトウェア開発支援システムに関するものである。

【0002】

【従来の技術】 近年の電子技術の発展により、家電機器などの組み込み機器においてもマイクロコンピュータなどプロセッサを組み込み、プログラム開発を伴った機器開発が行われている。加えて、組み込み機器の高機能化

2

に伴い、プログラムが大規模になり、プログラムの開発および保守の効率化のため、C言語など高級言語によるプログラム開発が行われている。

【0003】 また、プログラムの大規模化に従い、大容量のメモリがプログラムのコード格納やデータ格納のために必要とされている。さらに、プログラムの処理の高速化のため、メモリアccessを高速にするキャッシュメモリ搭載のプロセッサが組み込み機器に用いられている。図3に、キャッシュメモリ搭載のプロセッサのシステム構成の一例を示す。図3において、点線で囲まれた部分がプロセッサであり、601はCPU（中央演算装置）であり各種演算を行う。602は、プロセッサの外部に存在する主記憶装置であり、プログラム実行コード（オブジェクトコード）およびデータを記憶する。603はキャッシュメモリであり、主記憶装置へのアクセスを高速化するために機能する。604はバスであり、このバスを用いて、CPU601、主記憶装置602、キャッシュメモリ603の間でデータが転送される。

【0004】 このようなプロセッサにおける、キャッシュメモリの機能について説明する。主記憶装置602上のアドレスに対する読み込みアクセスの場合には、そのアドレスおよびそのデータがキャッシュメモリ603の対応するアドレス上に登録されていれば（キャッシュヒットと呼ぶ）、そのキャッシュメモリ603上のデータが参照され、そのアクセスが高速に実行される。

【0005】 また、アドレスおよびデータがキャッシュメモリ603上に未登録であれば（キャッシュミス）、主記憶装置602上のデータが読み込まれると同時に、アドレスとデータがキャッシュメモリ603上に登録される。この場合のアクセス時間は、はキャッシュヒット時に比べて大きい。同様に、主記憶装置602上のアドレスに対する書き込みアクセスの場合には、そのアドレスおよびそのデータがキャッシュメモリ603の対応するアドレス上に登録されていれば（キャッシュヒット）、そのキャッシュメモリ603上のデータのみが更新されることにより、そのアクセスが高速に実行される。

【0006】 また、そのアドレスおよびデータがキャッシュメモリ603上に未登録であれば（キャッシュミス）、主記憶装置602上のデータおよびそのアドレスとデータがキャッシュメモリ603上に登録されると同時にキャッシュメモリ603上のデータのみが更新される。この場合のアクセス時間は、キャッシュヒット時に比べて大きい。

【0007】 なお、主記憶装置上のアドレスとキャッシュメモリ上への対応づけの方式には、主記憶装置上の1つのアドレスに1つのキャッシュメモリ上のアドレスが対応づくダイレクトマップ方式や、2つのキャッシュメモリ上のアドレスが対応づく2ウェイセットアソシエティブ方式など幾つかの方式がある。また、キャッシュメモリの方式によってキャッシュメモリへの書き込みと

10

20

30

40

50

3

同時につねに主記憶装置上へも書きこみが行われる方式（ライトスルー方式）もある。

【0008】以上説明したようなキャッシュメモリ603を搭載したプロセッサにおいて、実行されるプログラムの性能評価および性能向上は、プログラム実行時のキャッシュヒット、キャッシュミス（キャッシュミス）の状況を考慮し行う必要がある。加えて、組み込み機器開発においては、リアルタイム応用など高速処理実現と低コスト実現の両立が求められ、より正確な性能評価およびそれに基いた性能向上が求められる。

【0009】しかし、従来は、性能評価として、実機での実行時間の測定や、またシミュレーションによるサイクル数の見積もりなどが行われてきた。また、キャッシュ搭載のプロセッサを対象とする場合は、性能評価の指標としてキャッシュのヒット率つまりアクセス回数に対するキャッシュヒットの比率が専ら用いられてきた。

【0010】このような性能評価を行い、速度要求を満たしていないプログラムのサブルーチンなど特定部分については、利用者がプログラムを変更し、速度の改善を図ってきた。プログラム変更の一つの方法は、キャッシュヒット率を向上させ、メモリアクセスの速度を向上させることであり、例えば、キャッシュヒット率が向上するように、プログラムで使用するデータの割付けアドレスを変更することである。

【0011】

【発明が解決しようとする課題】しかしながら、従来の方法においてキャッシュヒット率だけではその詳細な状況は判らず、また、シミュレーションにおけるプログラム動作のトレース結果では、トレースされる対象がアドレスだけであり、実際に利用者が変更する対象である高級言語レベルのプログラムには対応しておらず、性能向上のためのプログラム変更が困難であるという問題を有していた。

【0012】この発明は、上記の課題に鑑み、キャッシュメモリへのアクセスの状況を高級言語レベルで検出し、キャッシュメモリのヒット率が向上するようにシンボルの割付情報を生成することにより、キャッシュメモリ搭載プロセッサ上のプログラム開発を支援するソフトウェア開発支援システムを提供することを目的とする。また、キャッシュミスの分類の1つとして、キャッシュメモリの1つのエントリに複数の主記憶上のブロック（アドレス）が対応するために起因する競合性ミスがある。このキャッシュミスは、複数の主記憶上のブロックが1つのエントリに対応づかないようにプログラムを変更してやれば解決できる可能性があるキャッシュミスである。

【0013】この発明は、したがって上記の課題に鑑み、キャッシュミスのうち特に競合性ミスを検出し競合しているシンボルを競合しないようにシンボルの割付情報を生成することにより、キャッシュメモリ搭載プロセ

4

ッサ上のプログラム開発を支援するソフトウェア開発支援システムを提供することを目的とする。さらにこの発明は、上記の課題に鑑み、キャッシュメモリへのアクセスの状況を高級言語レベルで利用者に表示し、キャッシュメモリ搭載プロセッサ上のプログラムを支援するソフトウェア開発支援システムを提供することを目的とする。

【0014】

【課題を解決するための手段】請求項1記載のソフトウェア開発支援システムは、キャッシュメモリを搭載するプロセッサ上で実行されるプログラムのソフトウェア開発支援システムであって、プロセッサの動作をシミュレーションするシミュレーション手段と、このシミュレーション手段からキャッシュメモリへのアクセスに関する情報を採取するキャッシュアクセス情報採取手段と、プログラムのシンボル情報を管理するシンボル情報管理手段と、プログラムのシンボルの配置を変更するためのシンボル配置変更情報を生成するシンボル配置変更情報生成手段とを備えたものである。

【0015】請求項1記載のソフトウェア開発支援システムによれば、キャッシュアクセス情報採取手段で採取された情報と、シンボル情報管理手段で管理されているシンボル情報から、プログラムのメモリアクセス時の速度が高速になるようにシンボルの配置を変更するためのシンボルの配置変更情報を生成することができる。これにより、キャッシュメモリへのアクセスの状況が対象プログラムが記述される高級言語レベルで検知可能となり、アクセス速度を低下させるシンボルの配置を変更し、プログラムの実行性能を向上させることが可能となる。

【0016】請求項2記載のソフトウェア支援開発システムは、請求項1において、キャッシュアクセス情報採取手段が、キャッシュメモリでのエントリの競合性ミスに関する情報を採取するものである。請求項2記載のソフトウェア支援開発システムによれば、請求項1と同様な効果のほか、キャッシュアクセス情報採取手段で採取された情報からキャッシュ競合性ミスを検出し、シンボル情報と関連づけて、シンボルの配置変更情報をシンボル配置変更情報生成手段で生成することができる。これにより、キャッシュメモリ搭載のプログラムを開発する際に、キャッシュメモリへのアクセスの状況特に競合性ミスをシンボルレベルないし高級言語レベルで検出し、キャッシュメモリで競合性ミスが発生するシンボルの配置を変更する情報を生成するなど、その競合を解消するためにシンボルの配置を変更することが容易にでき、プログラム開発の効率化が図れる。

【0017】請求項3記載のソフトウェア支援開発システムは、請求項1または請求項2において、シンボル配置変更情報生成手段が、キャッシュメモリでのエントリの競合性ミスを低減するための情報を生成するものであ

5

る。請求項3記載のソフトウェア支援開発システムによれば、請求項1または請求項2と同様な効果のほか、競合性のキャッシュミスが発生したシンボルをキャッシュされていない空のエントリに対応するアドレスに変更するための情報を生成するなど、キャッシュメモリの競合性ミスを低減させるようなシンボルの配置変更情報が生成でき、より一層のプログラムの性能向上を図れる。

【0018】請求項4記載のソフトウェア支援開発システムは、キャッシュメモリを搭載するプロセッサ上で実行されるプログラムのソフトウェア支援開発システムであって、プロセッサの動作をシミュレーションするシミュレーション手段と、このシミュレーション手段からキャッシュメモリへのアクセスに関する情報を採取するキャッシュアクセス情報採取手段と、プログラムのシンボル情報を管理するシンボル情報管理手段と、シンボル情報を用いてキャッシュアクセスの情報を表示する表示手段とを備えたものである。

【0019】請求項4記載のソフトウェア支援開発システムによれば、キャッシュアクセス情報採取手段で採取された情報と、シンボル情報管理手段で管理されているシンボル情報を表示手段により関連付けて表示することができるので、キャッシュメモリへのアクセスの状況が、対象プログラムが記述される高級言語レベルで表示可能となり、評価性能に基いたプログラムの変更が容易になる。したがって、キャッシュメモリ搭載のプログラムを開発し評価する際に、キャッシュメモリへのアクセス状況を高級言語レベルで知ることができ、プログラム開発の効率化が図れる。

【0020】請求項5記載のソフトウェア支援開発システムは、請求項4において、キャッシュアクセス情報採取手段が、アクセス対象のアドレスおよびアクセス時のプログラムカウンタ値、スタックポインタ値ならびにアクセスがヒットかどうかの情報を採取するものである。請求項5記載のソフトウェア支援開発システムによれば、請求項4と同様な効果がある。

【0021】請求項6記載のソフトウェア支援開発システムは、請求項4または請求項5において、表示手段が、キャッシュメモリでのエントリ別に分類して表示するものである。請求項6記載のソフトウェア支援開発システムによれば、請求項4または請求項5と同様な効果がある。

【0022】

【発明の実施の形態】以下、この発明の実施の形態について説明する。この発明の第1の実施の形態を図1から図16により説明する。図1は、この発明の第1の実施の形態におけるソフトウェア支援開発システムに適用されるシステムの一例であり、高級言語によるソフトウェア開発およびそのシミュレーション、ならびにデバッグに係る全体的なシステムである。

【0023】図1において、105は、開発の対象とな

6

るアプリケーションを高級言語で記述したソースプログラムである。101はコンパイラであり、ソースプログラム105を入力し、対象とするプロセッサに対応したオブジェクトコード106と、デバッグ情報107を出力する。

【0024】102は、ソフトウェア開発支援システムであり、デバッグ103と、対象プロセッサの動作をシミュレーションするシミュレーション手段であるシミュレータ104とから構成される。開発支援システム102は、オブジェクトコード106とデバッグ情報107を入力し、アプリケーションのシミュレーションおよび高級言語レベルでのデバッグを実現する。

【0025】図9に、開発支援システム102が対象とするプロセッサの構成図を示す。図9は図3と比べて、キャッシュメモリが命令キャッシュメモリ1303とデータキャッシュメモリ1304に別れている点が異なる。よって、命令キャッシュメモリ1303とデータキャッシュメモリ1304についてのみ説明する。命令キャッシュメモリ1303は、命令の読み込み動作（フェッチ動作）時に、用いられるキャッシュメモリであり、データキャッシュメモリ1304は、データの読み込み動作（データリード動作）時とデータの書きこみ動作（データライト動作）時に用いられるキャッシュメモリである。

【0026】また、命令キャッシュメモリ1303は、内部にタグアレイ1306およびデータアレイ1307を備え、同様にデータキャッシュメモリ1304は、タグアレイ1308とデータアレイ1309を備える。タグアレイ1306、1308は、主記憶装置1302上のアドレスとキャッシュメモリのデータアレイ1307、1309上のアドレスを対応付けるタグ情報を保持する。また、データアレイ1307、1309は、上記のように対応付けられたデータの内容自体を保持する。

【0027】図4は、プロセッサのCPUの構成とともに、メモリとの関係を示す。図4において、701はCPUであり、プログラム実行時に主にデータを格納するデータレジスタ（D0、D1、D2、D3）703、704、705、706と、主にアドレスを格納するアドレスレジスタ（A0、A1、A2、A3）707、708、709、710と、演算結果フラグを格納するプロセッサ状態語レジスタ（PSW）711と、プログラム実行アドレスを指し示すプログラムカウンタレジスタ（PC）と、スタック領域のアドレスを指し示すスタックポインタレジスタ（SP）と、各種演算を行う演算器714から構成される。

【0028】702は、メモリ領域の概念図である。702に示すように、メモリ領域は、プログラム自体である実行コードを格納する実行コード格納領域715と、主にグローバルなデータを格納するデータ領域716と、主にローカルなデータを格納するスタック領域71

7

7から構成される。このメモリ領域には、すべてアドレスが割振られ、CPU701からはアドレスによりアクセスが行われる。この構成において、プロセッサは、実行コード格納領域715に格納されている命令を読み込み、読み込んだ命令に従って各種レジスタあるいはメモリ上のデータを用いて演算を行う。

【0029】このようなプロセッサでは、プロセッサ(CPU)とメモリ(主記憶)の間のデータの転送は、(1)メモリからプロセッサへの命令の読み込み(フェッチ)、(2)メモリからプロセッサへのデータの読み込み(データリード)、(3)プロセッサからメモリへのデータの書きこみ(データライト)の3種類に区分される。

【0030】表1に、ソースプログラムの一例を示す。

【0031】

【表1】

```
01:  int x, y;
02:
03:  void main() {
04:      int i;
05:      x = y = 0;
06:      for (i=0; i<100; i++) {
07:          x = i*i - i;
08:          y = y + x*2;
09:      }
10: }
```

```
01:  _TEXT  SECTION
02:
03:  0x100    add    -4, SP
04:  0x104    clr    D0
05:  0x108    mov    D0, (0x410)
06:  0x10C    mov    D0, (0x400)
07:  0x110    mov    D0, (SP)
08:  0x114    jmp    0x150
09:
10:  0x118    mov    (SP), D0
11:  0x11C    mov    D0, D1
12:  0x120    mul    D1, D0
13:  0x124    mov    (SP), D1
14:  0x128    sub    D1, D0
15:  0x12C    mov    D0, (0x400)
16:  0x130    mov    D0, D1
17:  0x134    mov    (0x410), D0
18:  0x138    add    D1, D0
19:  0x13C    add    D1, D0
20:  0x140    mov    D0, (0x410)
21:  0x144    mov    (SP), D0
22:  0x148    inc    D0
23:  0x14C    mov    D0, (SP)
24:
25:  0x150    mov    (SP), D0
26:  0x154    cmp    100, D0
27:  0x158    bge    0x160
28:  0x15C    jmp    0x118
29:
30:  0x160    ret
31:
32:
33:  _DATA  SECTION
34:  0x400    DS      4
35:  0x410    DS      4
```

【0035】表2では、説明のため機械語命令と対応するアセンブラ記述を示しているが、本質的には前記の2つは同一の情報を表現する。また、全ての機械語命令は、4バイトの長さのビットパターンである。なお、表2において、各行の左端の『数字:』は、説明のために

8

*【0032】表1に示すソースプログラムは、高級言語の一つであるC言語によって記述されている。なお、表1において、各行の左端の『数字:』は、説明のために付与した行番号である。表1のソースプログラムにおいて、1行目は、グローバル変数でint(整数)型変数xとyの宣言を示す。3行目から10行目までが、関数mainの定義部分である。関数mainでは、4行目に示すように、ローカル変数でint(整数)型変数iを用いて、6行目から9行目に示すように、変数iを0から99まで増加させて、7、8行目に示す演算を繰り返し行う処理が記述されている。

【0033】表2に、オブジェクトコードに対応するアセンブラ記述による説明図を示す。オブジェクトコードは、対象とするプロセッサの機械語命令で表現され、アドレスとそのアドレスに格納する値が情報として記述される。

【0034】

【表2】

付与した行番号であり、左端2番目の16進数による表記は、その命令あるいはデータが格納されるアドレスを示す。例えば、3行目の命令『add -4, SP』に対応するオブジェクトコードは、アドレス0x100番地に格納され、33行目に示すデータxに対応するデー

タは、アドレス0x400番地に格納される。

【0036】表2において、1行目から30行目までは、実行コード部分である。1行目は、アセンブラ記述での擬似命令であり、1行目に続く部分が実行コードであることを示す。また、33行目から35行目は、データ領域部分であり、33行目は、以下に続く部分がデータ領域であることを示すアセンブラ擬似命令である。34行目は、アドレス0x400番地以降に4バイト分の領域を割当ることを示す。同様に、35行目は、アドレス0x410番地以降に4バイト分の領域を割当ることを示す。

【0037】後述するシンボル情報によって、アドレス0x100番地がC言語プログラム上の関数mainの先頭番地に対応付けられ、また、アドレス0x400番地が変数xに、同様にアドレス0x410番地が変数yに対応付けられる。下記に3行目から30行目に記述されている実行コードで用いられているアセンブラ記述とプロセッサの命令の対応を示す。

【0038】(1) add 『加算データ』, 『被加算データ』

『加算データ』で示されるレジスタあるいはメモリの内容あるいは即値を『被加算データ』で示されるレジスタの内容に加算して、結果を『被加算データ』で示されるレジスタに格納する。例えば、3行目の記述は、即値-4をスタックポインタレジスタ(SP)の内容に加算して、加算結果をスタックポインタレジスタに格納する命令である。また、18行目の記述は、データレジスタD0の内容をデータレジスタD1の内容に加算して、加算結果をデータレジスタD1へ格納する命令である。

【0039】(2) clr 『対象レジスタ』

『対象レジスタ』で示されるレジスタの内容を値0にする。例えば、4行目の記述は、データレジスタD0の内容を0にする命令である。

(3) inc 『対象レジスタ』

『対象レジスタ』で示されるレジスタの内容を1増加して格納する。例えば、22行目の記述は、データレジスタD0の内容を1増加して格納する命令である。

【0040】(3) sub 『減算データ』, 『被減算データ』

『減算データ』で示されるレジスタあるいはメモリの内容あるいは即値を『被減算データ』で示されるレジスタの内容から減算して、結果を『被減算データ』で示されるレジスタに格納する。例えば、14行目の記述は、データレジスタD1の内容をデータレジスタD0の内容から減算して、減算結果をデータレジスタD0に格納する命令である。

【0041】(4) cmp 『減算データ』, 『被減算データ』

『減算データ』で示されるレジスタあるいはメモリの内容あるいは即値を『被減算データ』で示されるレジスタ

の内容から減算して、結果のフラグ(結果が正である、結果が負である、結果が0である)をプロセッサ状態語レジスタPSWに反映する。sub命令と異なり、減算結果は、レジスタには格納されない。例えば、26行目の記述は、データレジスタD0から即値100を減算して、その結果フラグをプロセッサ状態語レジスタPSWに反映する命令である。

【0042】(5) mov 『転送元データ』, 『転送先データ』

『転送元データ』で示されるレジスタあるいはメモリの内容あるいは即値を『転送先データ』で示されるレジスタあるいはメモリへ転送する。例えば、5行目の記述は、データレジスタD0の内容をアドレス0x410番地から始まる1ワード(4バイト)分のメモリ上の領域に転送する命令である。また、例えば、10行目の記述は、スタックポインタレジスタSPの内容が指し示すメモリ上の1ワード分の領域の内容を、データレジスタD0へ格納する命令である。

【0043】(6) jmp 『ジャンプ先』

『ジャンプ先』で示されるアドレスへ、プログラム実行の制御を変更する。例えば、8行目の記述は、0x150番地(25行目)にプログラム実行の制御を変更する命令である。なお、このようなジャンプ命令以外は、プログラム実行は逐次的に行われる。例えば、0x100番地(3行目)の命令の次は0x104番地(4行目)の命令が実行される。

【0044】(7) mul 『乗算データ』, 『被乗算データ』

『乗算データ』で示されるレジスタの内容を、『被乗算データ』で示されるレジスタの内容に乗じて、乗算結果を『被乗算データ』で示されるレジスタに格納する。例えば、12行目の記述は、データレジスタD1の内容をデータレジスタD0の内容に乗算して、その乗算結果をデータレジスタD0に格納する命令である。

【0045】(8) bge 『ジャンプ先』

プロセッサ状態語レジスタのフラグが正あるいは0の場合に、『ジャンプ先』で示されるアドレスへ、プログラム実行の制御を変更する。例えば、27行目の記述は、フラグが正あるいは0の場合に、0x160番地(30行目)にプログラム実行の制御を変更する命令である。

【0046】(9) ret

サブルーチン呼び出し元へプログラムの制御を変更する。しかし、オブジェクトコードには、前記に説明したようなC言語での関数あるいは変数との対応付けの情報はなく、実際には次に説明するシンボル情報で示される。表3に、デバッグ情報の一例であるシンボル情報を示す。

【0047】

【表3】

11

シンボル情報

```

01: 関数 main
02:   開始アドレス 0x100
03:   引数          void
04:   戻り値        void
05:   ローカル変数
06:     int i      (SP)
07:
08:
09: 変数 x
10:   開始アドレス 0x400
11:   int型
12:
13: 変数 y
14:   開始アドレス 0x410
15:   int型

```

【0048】なお、表3において、各行の左端の『数字:』は、説明のために付与した行番号である。シンボル情報では、高級言語上の変数名や関数名などシンボルとオブジェクトコード上でのアドレスとの対応付けを行う情報である。表3において、1行目から6行目では、シンボル“main”が関数であり、その開始アドレスが0x100番地であり、関数の引数および戻り値がvoid型であり、加えて関数main内のローカル変数iがスタックポインタ(SP)レジスタが指すアドレスに割付られており、その型がint(整数)型であることを示している。

【0049】また、9行目から11行目までには、シンボル“x”がグローバル変数であり、その開始アドレスが0x400番地であり、その型がint(整数)型であることを示している。同様に、13行目から15行目では、シンボル“y”がグローバル変数であり、その開始アドレスが0x410番地であり、その型がint(整数)型であることを示している。

【0050】図2に、前記開発支援システム102の構成の一例を示す。図2において、201はデバッグ103であり、デバッグ操作処理部203、実行制御部204、オブジェクトコード管理部205、シンボル情報管理手段であるシンボル情報管理部206、キャッシュアクセス情報採取手段であるキャッシュアクセス情報採取部207、シンボル配置変更情報生成手段であるシンボル配置変更情報生成部208を備える。

【0051】また、202はシミュレータ104であり、CPUをシミュレーションするCPUシミュレーション部209、内蔵メモリや外付けメモリをシミュレーションするメモリシミュレーション部210、キャッシュメモリをシミュレーションするキャッシュメモリシミュレーション部211、シミュレーション実行の時刻を管理計測する時刻管理部212を備える。

【0052】デバッグ操作処理部203は、オブジェクトコード213のロードや実行開始、ブレークアドレス設定などユーザからのデバッグコマンドの入力を受付

12

け、処理する手段である。例えば、オブジェクトコード213のロードコマンド処理時には、オブジェクトコード管理部205から、オブジェクトコード213を読み込み、シミュレータ202内のメモリシミュレーション部210にオブジェクトコード213の内容を書き込む処理を行う。

【0053】また、実行開始コマンド処理時には、シミュレータ203内のCPUシミュレーション部209のプログラムカウンタ(PC)レジスタに開始アドレスを設定し、実行制御部204により、オブジェクトコードのシミュレーションの実行を開始する。実行制御部204は、シミュレータ202を用いて対象オブジェクトコードの実行を実現する。シミュレーションの実行は、命令を単位として制御可能である。なお、これはプロセッサの動作サイクル毎の制御でもよい。

【0054】この実行制御部204では、ユーザに指定されたブレークポイントにオブジェクトコードの実行が到達するまで、あるいは既定の停止アドレス(例えばexit関数のアドレス)に実行が到達するまで、オブジェクトコード213の実行が繰り返し行われる。図5に、実行制御部204の動作フローチャートを示す。図5において、ステップ801では、シミュレータ202からCPU701(1301)内のプログラムカウンタレジスタ(PC)の値を読み出し、実行アドレスが既定の停止アドレス(例えばexit関数)に到達したかを判定する。到達していれば処理を終了し、到達していなければステップ802に進む。

【0055】ステップ802では、デバッグ操作処理部203から利用者が指定したブレークアドレスの情報を読み出し、実行アドレスがブレークアドレスに到達したかを判定する。到達していれば処理を終了し、到達していなければステップ803に進む。ステップ803では、シミュレーション部202を用いて1命令分のシミュレーションを行う。

【0056】図2において、209はCPUシミュレーション部であり、CPU701(1301)の動作をシミュレーションする。つまり、実行する命令をプログラムカウンタ(PC)レジスタが指し示すアドレスから読み取り(以下フェッチ動作とよぶ)、読み込んだ命令に従ってレジスタあるいはメモリ上のデータを用いて演算を行う。

【0057】フェッチ動作をシミュレーションする場合、メモリシミュレーション部210にアドレス(フェッチアドレス)を入力して、そのアドレス上のデータを得る。また、演算のシミュレーションにおいて、入力としてメモリ上のデータを読み込む(データリード)際には、メモリシミュレーション部210にアドレスを入力し、そのアドレス上のデータを得る。

【0058】同様に演算時に結果をメモリ上に出力する(データライト)際には、メモリシミュレーション部2

10へアドレスと出力するデータ内容を入力し、そのアドレス上のデータへ書き込む。図6に、CPUシミュレーション部209の1命令分のシミュレーションを行う場合の動作フローチャートを示す。図6において、ステップ901では、CPUシミュレーション部209のプログラムカウンタ(PC)レジスタが指し示すアドレスと、1命令の語長である4バイトをメモリシミュレーション部210に指定して、そのアドレス上のデータを読み込む。また、同時に命令の読み込みに必要なサイクル数を求める。

【0059】ステップ902では、入力した機械語命令である4バイト長のビットパターンを解析し、命令の種類および命令実行に必要な入力先および出力先を求める。ビットパターンには、入力元あるいは出力先のレジスタの指定番号や、即値あるいはメモリ上のアドレスが埋込まれている。ステップ903では、命令実行に必要なデータを入力する。データの入力元は命令によって異なり、レジスタの内容やメモリ上の内容あるいは即値である。入力元がレジスタの内容である場合は、CPUシミュレーション部209の中のレジスタ内容管理領域を参照することによって、入力値を得る。また、入力元がメモリ上の内容の場合は、メモリシミュレーション部210に入力元アドレスおよびデータ長を指定することにより、入力値を得る。また、同時にデータの読み込みに必要なサイクル数を求める。また、即値の場合は、ステップ902で解析した結果を参照する。

【0060】ステップ904では、ステップ902で解析した結果を用いて命令の種類を判別し、命令の種類に応じてステップ905a、905b、905c、905d、905e、905f、905g、905h、905iに進む。命令がadd命令の場合は、ステップ905aに進む。ステップ905aでは、add命令の演算シミュレーションを行う。つまりステップ903で入力したデータを用いて加算を実行し、演算結果を得る。

【0061】同様に、命令がinc命令の場合は、ステップ905bに進む。ステップ905bでは、inc命令の演算シミュレーションを行う。つまりステップ903で入力したデータと値1の加算を実行し、演算結果を得る。同様に、命令がsub命令の場合は、ステップ905cに進む。ステップ905cでは、sub命令の演算シミュレーションを行う。つまりステップ903で入力したデータを用いて減算を実行し、演算結果を得る。

【0062】同様に、命令がcmp命令の場合は、ステップ905dに進む。ステップ905dでは、cmp命令の演算シミュレーションを行う。つまりステップ903で入力したデータによる減算を実行し、演算結果フラグを得る。同様に、命令がmul命令の場合は、ステップ905eに進む。ステップ905eでは、mul命令の演算シミュレーションを行う。つまりステップ903で入力したデータを用いて乗算を実行し、演算結果を得

る。

【0063】同様に、命令がclr命令の場合は、ステップ905fに進む。ステップ905fでは、clr命令の演算シミュレーションを行う。つまり演算結果として0を得る。同様に、命令がmov命令の場合は、ステップ905gに進む。ステップ905gでは、mov命令の演算シミュレーションを行う。つまりステップ903で入力したデータを演算結果とする。

【0064】同様に、命令がjmp命令の場合は、ステップ905hに進む。ステップ905hでは、jmp命令の演算シミュレーションを行う。つまりステップ902での解析結果からジャンプ先のアドレスを入力して、CPUシミュレーション部209のプログラムカウンタレジスタに格納する。同様に、命令がbge命令の場合は、ステップ905iに進む。ステップ905iでは、bge命令の演算シミュレーションを行う。つまりCPUシミュレーション部209からプロセッサ状態語レジスタPSWの値を読み出し、フラグを参照して、条件が成立している場合(演算結果フラグが正あるいは0)に、ステップ902での解析結果からジャンプ先のアドレスを入力して、CPUシミュレーション部209のプログラムカウンタレジスタに格納する。

【0065】ステップ905a、905b、905c、905d、905e、905f、905gでは、次に処理終了後ステップ906に進む。ステップ905h、905iでは、次に処理終了後ステップ907に進む。ステップ905a、905b、905c、905d、905e、905f、905g、905hおよび905iでは、それぞれ命令の実行にかかるサイクル数を求める。これは、命令の種類およびその実行時のCPUの状態により決定され、求まる。

【0066】ステップ906では、CPUシミュレーション部209のプログラムカウンタレジスタの値を入力し、値4を加算して格納する。これによりプログラム実行が逐次的に行われる。ステップ907では、ステップ905a、905b、905c、905d、905e、905f、905gで得られた演算結果を、ステップ902での解析で判別した出力先に格納する。演算結果の出力先は、命令によって異なり、レジスタやメモリ上の領域である。出力先がレジスタである場合は、CPUシミュレーション部209の中のレジスタ内容管理領域へ格納する。また、出力先がメモリ上の領域の場合は、メモリシミュレーション部210に出力先アドレスと出力データおよび出力データ長を指定することにより、出力を行う。

【0067】ステップ908では、命令の読み込みにかかるサイクル数(時間)、命令の実行に必要なとなるデータの入力にかかるサイクル数、命令の実行にかかるサイクル数、演算結果の格納にかかるサイクル数など1命令実行に必要なサイクル数を全て算出し、時刻管理部212

へ出力する。図2における時刻管理部212は、1命令ごとのサイクル数を合計していくことにより、実行プログラムのサイクル数を求める。

【0068】図7にメモリシミュレーション部210の動作フローチャートを示す。図7において、ステップ1001では、メモリシミュレーション部210へのアクセスの種別の判定を行う。前述したようにCPUシミュレーション部209からは、命令の読み込み動作（フェッチ動作）の要求、データの読み込み動作（データリード動作）の要求およびデータの書き込み動作（データライト動作）の要求がくる。また、デバグ201からは、デバグコマンドでのメモリ参照操作の場合に、データの読み込み（ダイレクトリード動作）の要求が、またオブジェクトコード213のロード処理やデバグ201上で*

*のメモリ変更操作処理の場合に、データの書き込み（ダイレクトライト動作）の要求がくる。このダイレクトリード動作あるいは、ダイレクトライト動作の場合は、プロセッサのシミュレーションは行わない。つまりプロセッサの状態を変更せず、またサイクル数を算出しない。

【0069】フェッチ動作の場合は、ステップ1002で、アクセス対象として指定されたアドレスがキャッシュ空間かどうかを判定する。この実施の形態で対象とするプロセッサは、アクセスの対象となるアドレスによってキャッシュが動作するかどうか違い、表4に示すような、区分となっている。

【0070】

【表4】

アドレス範囲	キャッシュ属性
0x00000000-0x3FFFFFFF	非キャッシュ空間
0x40000000-0x7FFFFFFF	キャッシュ空間
0x80000000-0xFFFFFFFF	非キャッシュ空間

【0071】表4は、メモリシミュレーション部210が保持するキャッシュ属性判定表である。表4は、アドレス0x00000000番地から0x3FFFFFFF番地までのアドレス空間およびアドレス0x80000000番地から0xFFFFFFFF番地までのアドレス空間が非キャッシュ空間であり、アドレス0x40000000番地から0x7FFFFFFF番地までのアドレス空間がキャッシュ空間であることを示している。キャッシュ空間の場合、その空間内のアドレスへのアクセスは全てキャッシュメモリを利用して高速化され、非キャッシュ空間の場合は、キャッシュメモリが利用されない。

【0072】ステップ1002において、指定アドレスがキャッシュ空間の場合は、ステップ1003に進む。そうでない場合はステップ1004に進む。ステップ1003では、キャッシュメモリシミュレーション部211に対してアドレスと読み込むデータ長を指定することにより、キャッシュメモリのフェッチ動作のシミュレーションを行う。

【0073】ステップ1004では、メモリシミュレーション部210内のメモリデータ格納領域を参照し、指定アドレスのメモリデータを得る。メモリデータ格納領域は図8に示すように、実際のメモリデータとそのアドレスを管理保持する領域である。図8において、1201は、アドレス0x00000000番地から0x00003FFF番地までのアドレス空間のデータ内容を保持管理する領域である。同様に、1202はアドレス0x40000000番地から0x40003FFF番地までのアドレス空間のデータ内容を、1203は0x80000000番地から0x80003FFF番地まで

のアドレス空間のデータ内容を保持管理する領域である。

【0074】ステップ1005では、アクセスにかかるサイクル数を算出する。キャッシュ空間の場合は、キャッシュメモリシミュレーション部211で算出されたサイクル数を参照する。また非キャッシュ空間の場合は、シミュレーション対象のメモリのウェイト数およびメモリとプロセッサ間のバス幅などの情報からサイクル数を算出する。

【0075】同様に、データリード動作の場合は、ステップ1006で、アクセス対象として指定されたアドレスがキャッシュ空間かどうかを判定する。指定アドレスがキャッシュ空間の場合は、ステップ1007に進む。そうでない場合は、ステップ1008に進む。ステップ1007では、キャッシュメモリシミュレーション部211に対してアドレスと読み込むデータ長を指定することにより、キャッシュメモリのデータリード動作のシミュレーションを行う。

【0076】ステップ1008では、ステップ1004と同様に、メモリシミュレーション部210内のメモリデータ格納領域を参照し、指定アドレスのメモリデータを得る。ステップ1009では、ステップ1005と同様に、アクセスにかかるサイクル数を算出する。

【0077】同様に、データライト動作の場合は、ステップ1010で、アクセス対象として指定されたアドレスがキャッシュ空間かどうかを判定する。指定アドレスがキャッシュ空間の場合は、ステップ1011に進む。そうでない場合は、ステップ1012に進む。ステップ1011では、キャッシュメモリシミュレーション部211に対してアドレスと書き込むデータとその

プ1613に進む。

【0087】ステップ1612では、取得したタグ情報と指定アドレスを比較し、キャッシュヒットかどうかを判定する。キャッシュミスと判定された場合は、競合性ミスであり、この場合競合性ミスであることを記憶する。なお、競合性ミスであることはステップ1615において、キャッシュアクセス情報採取部207に登録される。

【0088】キャッシュミスの場合、ステップ1613に進む。ステップ1613では、エントリにそのアドレスを登録する。ステップ1614では、データリード動作におけるデータキャッシュミスのサイクル数を求める。ステップ1615では、データキャッシュへのデータリード動作時のアクセスがキャッシュミスであったこと、ならびにアクセスアドレス、データ長、およびエントリをキャッシュアクセス情報採取部207に登録する。また、CPUシミュレーション部209からプログラムカウンタレジスタ(PC)の値と、スタックポインタレジスタ(SP)の値を取得し、共に登録する。また、競合性ミスであったかどうか登録する。

【0089】ステップ1616では、データリード動作におけるデータキャッシュヒット時のサイクル数を求める。ステップ1617では、データキャッシュへのデータリード動作時のアクセスがキャッシュヒットであったことをキャッシュアクセス情報採取部207に登録する。またアクセスアドレス、データ長、およびエントリをキャッシュアクセス情報採取部207に登録する。また、CPUシミュレーション部209からプログラムカウンタレジスタ(PC)の値と、スタックポインタレジスタ(SP)の値を取得し、共に登録する。

【0090】同様に、動作がデータライト動作の場合は、ステップ1618に進み、データキャッシュメモリ上のエントリを指定されたアドレスから算出する。ステップ1619では、エントリのタグ情報および有効フラグをデータキャッシュタグ情報管理部1403から検索し、有効フラグを判定する。そのエントリが有効な場合は、ステップ1620に進み、無効の場合はステップ1621に進む。

【0091】ステップ1620では、取得したタグ情報と指定アドレスを比較し、キャッシュヒットかどうかを判定する。キャッシュミスと判定された場合は、競合性ミスであり、この場合競合性ミスであることを記憶する。なお、競合性ミスであることはステップ1623において、キャッシュアクセス情報採取部207に登録される。

【0092】キャッシュミスの場合、ステップ1621に進む。ステップ1621では、エントリにそのアドレスを登録する。ステップ1622では、データライト動作におけるデータキャッシュミスのサイクル数を求める。ステップ1623では、データキャッシュへのデー

タライト動作時のアクセスがキャッシュミスであったこと、ならびにアクセスアドレス、データ長、およびエントリをキャッシュアクセス情報採取部207に登録する。また、CPUシミュレーション部209からプログラムカウンタレジスタ(PC)の値と、スタックポインタレジスタ(SP)の値を取得し、共に登録する。また、競合性ミスであったかどうか登録する。

【0093】ステップ1624では、データライト動作におけるデータキャッシュヒット時のサイクル数を求める。ステップ1625では、データキャッシュへのデータライト動作時のアクセスがキャッシュヒットであったことをキャッシュアクセス情報採取部207に登録する。また、アクセスアドレス、データ長、およびエントリをキャッシュアクセス情報採取部207に登録する。また、CPUシミュレーション部209からプログラムカウンタレジスタ(PC)の値と、スタックポインタレジスタ(SP)の値を取得し、共に登録する。

【0094】図13に、キャッシュアクセス情報採取部207のブロック図を示す。図13において、1701で示すキャッシュアクセス情報採取部207は、命令キャッシュアクセス情報管理部1702と、データキャッシュアクセス情報管理部1703を備える。命令キャッシュアクセス情報管理部1702は、フェッチアクセス情報1704を保持する。フェッチアクセス情報1704は、キャッシュメモリでのエントリ番号、フェッチアクセスにおけるアクセスアドレスとそのアクセスがキャッシュヒットであったかどうかの情報、そのデータ長、また対応するシンボルとそのアクセスの回数およびキャッシュミスの場合はその内の競合性ミスの回数を含む情報である。

【0095】また同様に、データキャッシュアクセス情報管理部1703は、データリードアクセス情報1705とデータライトアクセス情報1706を保持する。データリードアクセス情報1705は、データリードアクセスにおけるアクセスアドレスとそのアクセスがキャッシュヒットであったかどうかの情報、そのデータ長、また対応するシンボルとそのアクセスの回数、およびキャッシュミスの場合はその内の競合性ミスの回数を含む情報である。

【0096】同様に、データライトアクセス情報1706は、データライトアクセスにおけるアクセスアドレスとそのアクセスがキャッシュヒットであったかどうかの情報、そのデータ長、また対応するシンボルとそのアクセスの回数、およびキャッシュミスの場合はその内の競合性ミスの回数を含む情報である。図14に、キャッシュアクセス情報採取部207の動作フローチャートを示す。図14において、ステップ1801では、キャッシュメモリシミュレーション部1401から入力された採取情報の種別の判定を行う。採取情報が、フェッチ動作に関するものならばステップ1802に進み、データリ

ード動作に関するものならばステップ1804に進み、データライト動作に関するものならばステップ1806に進む。

【0097】ステップ1802では、フェッチアクセスの対象アドレスをシンボル情報から検索し、シンボルとの関連を付ける。すなわち、アドレスをキーにシンボル情報を検索し、シンボル名を得る。ステップ1803では、フェッチアクセス情報1704に、アクセス対象のアドレス、キャッシュヒットか否か、アクセスデータ長、対応付けられたシンボルを登録する。既に同一の情報
10 情報が登録されていればそのアクセスの回数の値を増やす。

【0098】ステップ1804では、データリードアクセスの対象アドレスをシンボル情報から検索し、シンボルとの関連を付ける。すなわち、アドレスをキーにシンボル情報を検索し、シンボル名を得る。また、その際にシンボル情報中で局所変数のように関数とスタックポインタレジスタ（SP）からのオフセットして定義されているシンボル情報は、前記の採取情報に含まれているアクセス時のプログラムカウンタ（PC）から関数が判別
20 でき、さらにアクセス時のスタックポインタレジスタ（SP）の値からシンボルとの対応付けができる。

【0099】ステップ1805では、データリードアクセス情報1705に、アクセス対象のアドレス、キャッシュヒットか否か、アクセスデータ長、対応付けられたシンボルを登録する。既に同一の情報が登録されていればそのアクセスの回数の値を増やす。同様に、ステップ1806では、データライトアクセスの対象アドレスをシンボル情報から検索し、シンボルとの関連を付ける。すなわち、アドレスをキーにシンボル情報を検索し、シン
30 ンボル名を得る。また、その際にシンボル情報中で局所変数のように関数とスタックポインタレジスタ（SP）からのオフセットして定義されているシンボル情報は、前記の採取情報に含まれているアクセス時のプログラム *
変数 $x: 0x0400 \rightarrow 0x1140$... (1)

のように表示する。

【0104】データキャッシュに関して同様に、ステップ1905から1908まで、それぞれステップ1901から1904に対応する処理を行う。ステップ1905において、ステップ1901と同様に、データキャッ
40 シュに関する情報であるデータリードアクセス情報1705およびデータライトアクセス情報1706について、各エントリの情報を走査し、図16（b）に示すデータキャッシュ競合情報表2102に登録する。ステップ1906において、データキャッシュ競合情報管理表2102から競合ミスをおこしたシンボルを検索する。シンボルがあればステップ1907に進み、シンボルがなければ処理を終了する。例えば、競合するシンボルとしては、『変数x』と『変数i@main』が得られ
50

*カウンタ（PC）から関数が判別でき、さらにアクセス時のスタックポインタレジスタ（SP）の値からシンボルとの対応付けができる。

【0100】ステップ1807では、データライトアクセス情報1706に、アクセス対象のアドレス、キャッシュヒットか否か、アクセスデータ長、および、対応付けられたシンボルを登録する。既に同一の情報が登録されていればそのアクセスの回数の値を増やす。なお、シンボル名が対応付かない場合は、シンボル名なしとして登録する。

【0101】図15に、プログラムのシンボルの配置を変更するためのシンボル配置変更情報生成部208の動作フローチャートを示す。ステップ1901において、命令キャッシュに関する情報であるフェッチアクセス情報1704について、各エントリの情報を走査し、競合性ミスが発生しなかったシンボル、競合性ミスが発生したシンボルをそれぞれ図16（a）に示す命令キャッシュ競合情報表2101に登録する。

【0102】ステップ1902において、命令キャッシュ競合情報管理表2101から競合ミスをおこしたシンボルを検索する。シンボルがあればステップ1903に進み、シンボルがなければステップ1905に進む。ステップ1903において、命令キャッシュ競合情報管理表2101からキャッシュがヒットしていない空のエントリを検索する。空のエントリがあればステップ1904に進み、空のエントリがなければステップ1905に進む。

【0103】ステップ1904において、競合ミスをおこしたシンボルを空のエントリ割付け、対応するアドレスを生成する。空のエントリ番号からアドレスの変更は、例えばある既定値からのエントリ番号に対応するオフセットを足し込むなどして求めることが可能である。生成する情報であるシンボル配置変更情報は、たとえば

【0105】ステップ1907において、データキャッシュ競合情報管理表2102からキャッシュがヒットしていない空のエントリを検索する。空のエントリがあればステップ1908に進み、空のエントリがなければ処理を終了する。例えば、空のエントリとして0x114が得られる。ステップ1908において、競合ミスをおこしたシンボルを空のエントリ割付け、対応するアドレスを生成する。

【0106】上記の表示（1）に、シンボル配置変更情報生成部208によるシンボル配置変更情報の一例を示す。表示（1）は、シンボルxをアドレス0x0400から0x1140へ変更することを示している。以上のように、この第1の実施の形態によれば、キャッシュメモリへのアクセスの状況を高級言語のシンボルとして検出し、キャッシュメモリで競合性ミスが発生するシンボ

ルの配置を変更する情報が生成できる。

【0107】この発明の第2の実施の形態を図17から図19により説明する。すなわち、このソフトウェア開発支援システムは、第1の実施の形態の図2に示すシンボル配置変更情報生成部208に代えて、表示手段であるキャッシュアクセス情報表示部216を設けたものである。図18に、キャッシュアクセス情報表示部216の動作フローチャートを示す。ステップ1901において、命令キャッシュに関する情報であるフェッチアクセス情報をエントリ別にわけて表示する。ステップ1902において、データキャッシュに関する情報であるデータリードアクセス情報とデータライトアクセス情報とをエントリ別にわけて表示する。

【0108】図19にキャッシュアクセス情報表示部216によるキャッシュアクセス情報の一例を示す。図19において、エントリおよびシンボル別にデータリード動作、データライト動作のヒットとミスの頻度の表示を示している。このように、キャッシュメモリへのアクセスの状況を高級言語レベルで利用者に表示することができる。

【0109】第2の実施の形態によれば、キャッシュアクセス情報採取部207で採取された情報と、シンボル情報管理部206で管理されているシンボル情報をキャッシュアクセス情報表示部216により関連付けて表示することができるので、キャッシュメモリへのアクセスの状況が、対象プログラムが記述される高級言語レベルで表示可能となり、評価性能に基いたプログラムの変更が容易になる。したがって、キャッシュメモリ搭載のプログラムを開発し評価する際に、キャッシュメモリへのアクセス状況を高級言語レベルで知ることができ、プログラム開発の効率化が図れる。

【0110】その他は、第1の実施の形態と同様である。なお、他の実施の形態として、第1の実施の形態に第2の実施の形態のキャッシュアクセス情報表示部216を付加したソフトウェア開発支援システムも可能である。また、この発明において、キャッシュアクセス情報を採取する区間を指定する手段を設け、指定された区間についてのみキャッシュアクセスに関する情報を採取することにより、高速性が求められるプログラムの特定の区間のみについて、性能向上を図ることも可能である。

【0111】

【発明の効果】請求項1記載のソフトウェア開発支援システムによれば、キャッシュアクセス情報採取手段で採取された情報と、シンボル情報管理手段で管理されているシンボル情報から、プログラムのメモリアクセス時の速度が高速になるようにシンボルの配置を変更するためのシンボルの配置変更情報を生成することができる。これにより、キャッシュメモリへのアクセスの状況が対象プログラムが記述される高級言語レベルで検知可能となり、アクセス速度を低下させるシンボルの配置を変更

し、プログラムの実行性能を向上させることが可能となる。

【0112】請求項2記載のソフトウェア開発システムによれば、請求項1と同様な効果のほか、キャッシュアクセス情報採取手段で採取された情報からキャッシュ競合性ミスを検出し、シンボル情報と関連づけて、シンボルの配置変更情報をシンボル配置変更情報生成手段で生成することができる。これにより、キャッシュメモリ搭載のプログラムを開発する際に、キャッシュメモリへのアクセスの状況特に競合性ミスをシンボルレベルないし高級言語レベルで検出し、キャッシュメモリで競合性ミスが発生するシンボルの配置を変更する情報を生成するなど、その競合を解消するためにシンボルの配置を変更することが容易にでき、プログラム開発の効率化が図れる。

【0113】請求項3記載のソフトウェア開発システムによれば、請求項1または請求項2と同様な効果のほか、競合性のキャッシュミスが発生したシンボルをキャッシュされていない空のエントリに対応するアドレスに変更するための情報を生成するなど、キャッシュメモリの競合性ミスを低減させるようなシンボルの配置変更情報が生成でき、より一層のプログラムの性能向上を図れる。

【0114】請求項4記載のソフトウェア開発システムによれば、キャッシュアクセス情報採取手段で採取された情報と、シンボル情報管理手段で管理されているシンボル情報を表示手段により関連付けて表示することができるので、キャッシュメモリへのアクセスの状況が、対象プログラムが記述される高級言語レベルで表示可能となり、評価性能に基いたプログラムの変更が容易になる。したがって、キャッシュメモリ搭載のプログラムを開発し評価する際に、キャッシュメモリへのアクセス状況を高級言語レベルで知ることができ、プログラム開発の効率化が図れる。

【0115】請求項5記載のソフトウェア開発システムによれば、請求項4と同様な効果がある。請求項6記載のソフトウェア開発システムによれば、請求項4または請求項5と同様な効果がある。

【図面の簡単な説明】

【図1】この発明の第1の実施の形態におけるソフトウェア開発支援システムに適用されるシステムのブロック図である。

【図2】この発明の第1の実施の形態におけるソフトウェア開発支援システムのブロック図である。

【図3】キャッシュメモリ搭載プロセッサのシステムブロック図である。

【図4】プロセッサの動作モデルを説明する説明図である。

【図5】図2の実行制御部204の動作フローチャートである。

【図6】図2のCPUシミュレーション部209の動作フローチャートである。

【図7】図2のメモリシミュレーション部210の動作フローチャートである。

【図8】メモリシミュレーション部のメモリデータ格納領域の説明図である。

【図9】この実施の形態におけるキャッシュメモリ搭載プロセッサのシステムブロック図である。

【図10】キャッシュシミュレーション部のブロック図である。

【図11】キャッシュタグ情報管理表を示す説明図である。

【図12】キャッシュシミュレーション部211の動作フローチャートである。

【図13】キャッシュアクセス情報採取部のブロック図である。

【図14】キャッシュアクセス情報採取部の動作フローチャートである。

【図15】シンボル配置変更情報生成部の動作フローチャートである。

【図16】キャッシュ競合情報表を示すブロック図である。

【図17】第2の実施の形態におけるソフトウェア開発支援システムのブロック図である。

【図18】キャッシュアクセス情報表示部の動作フローチャートである。

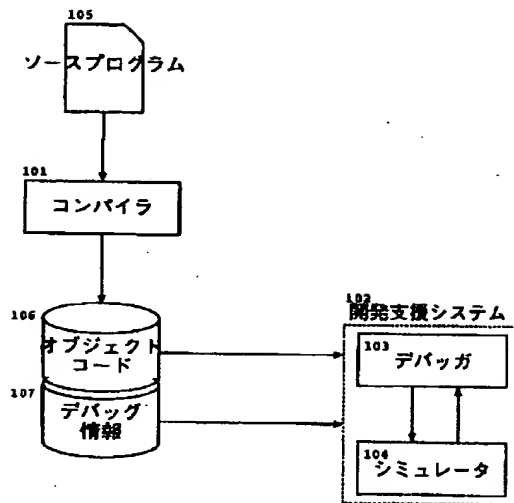
【図19】キャッシュアクセス情報の表示例を示す説明図である。

【符号の説明】

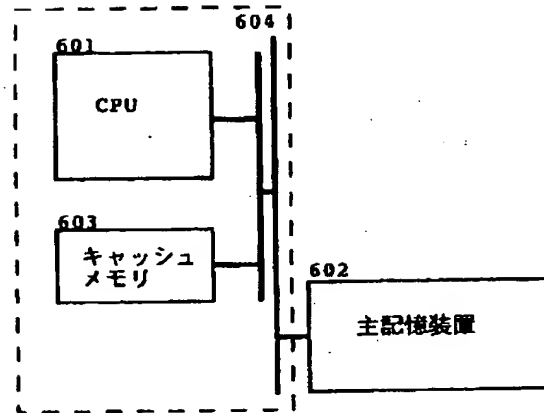
101 コンパイラ
102 開発支援システム
103 デバッガ
104 シミュレータ
105 ソースプログラム
106 オブジェクトコード
107 デバッグ情報
201 デバッガ
202 シミュレータ
203 デバッグ操作処理部
204 実行制御部
205 オブジェクトコード管理部
206 シンボル情報管理部
207 キャッシュアクセス情報採取部
208 シンボル配置変更情報生成部
209 CPUシミュレーション部
210 メモリシミュレーション部

211 キャッシュメモリシミュレーション部
212 時刻管理部
216 キャッシュアクセス情報表示部
601 CPU
602 主記憶装置
603 キャッシュメモリ
604 バス
701 CPU
702 メモリ
703、704、705、706 データレジスタ
707、708、709、710 アドレスレジスタ
711 プロセッサ状態語レジスタ
712 プログラムカウンタレジスタ
713 スタックポインタレジスタ
714 演算器
715 実行コード格納領域
716 データ格納領域
717 スタック領域
1201、1202、1203 アドレス空間メモリデータ格納領域
1301 CPU
1302 主記憶装置
1303 命令キャッシュメモリ
1304 データキャッシュメモリ
1305 バス
1306 命令キャッシュメモリのタグアレイ
1307 命令キャッシュメモリのデータアレイ
1308 データキャッシュメモリのタグアレイ
1309 データキャッシュメモリのデータアレイ
1401 キャッシュメモリシミュレーション部
1402 命令キャッシュタグ情報管理部
1403 データキャッシュタグ情報管理部
1501 有効フラグ
1502 エントリ
1503 タグ
1504 有効なエントリの一例
1505 無効なエントリの一例
1701 キャッシュアクセス情報採取部
1702 命令キャッシュアクセス情報管理部
1703 データキャッシュアクセス情報管理部
1704 フェッチアクセス情報
1705 データリードアクセス情報
1706 データライトアクセス情報
2101 命令キャッシュ競合情報管理表
2102 データキャッシュ競合情報管理表

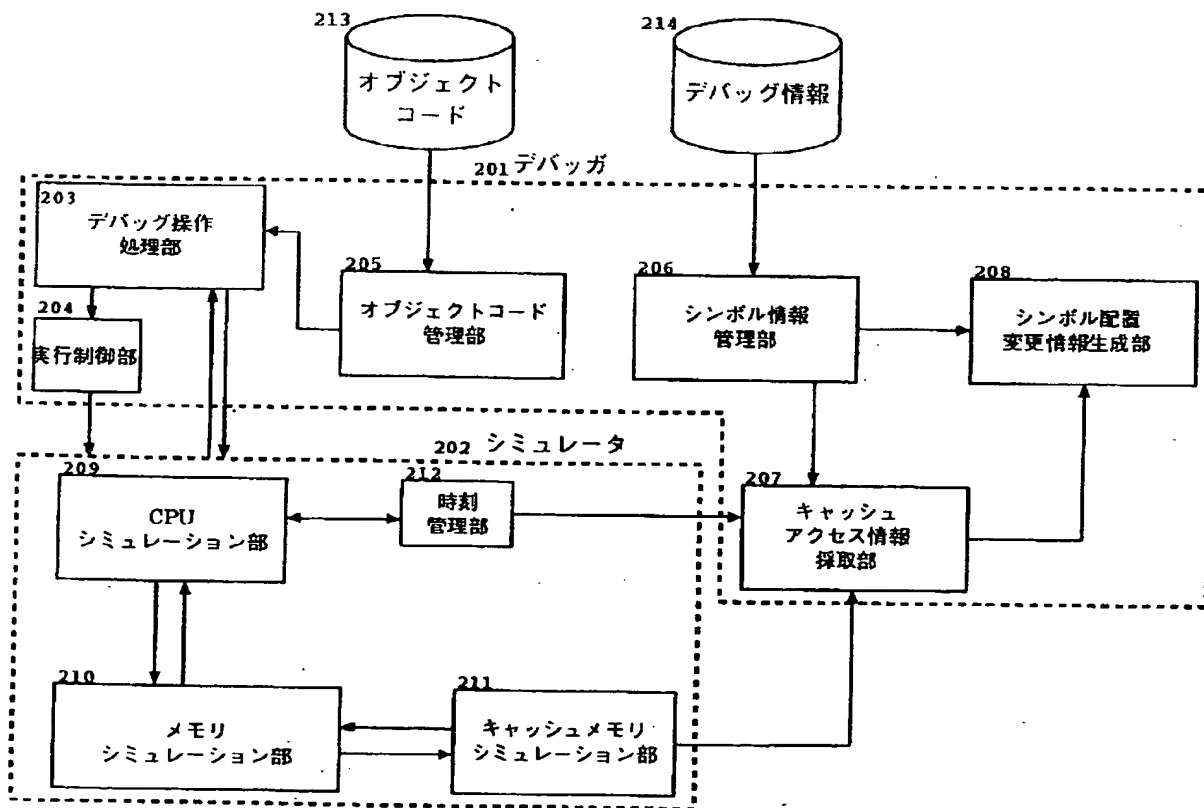
【図1】



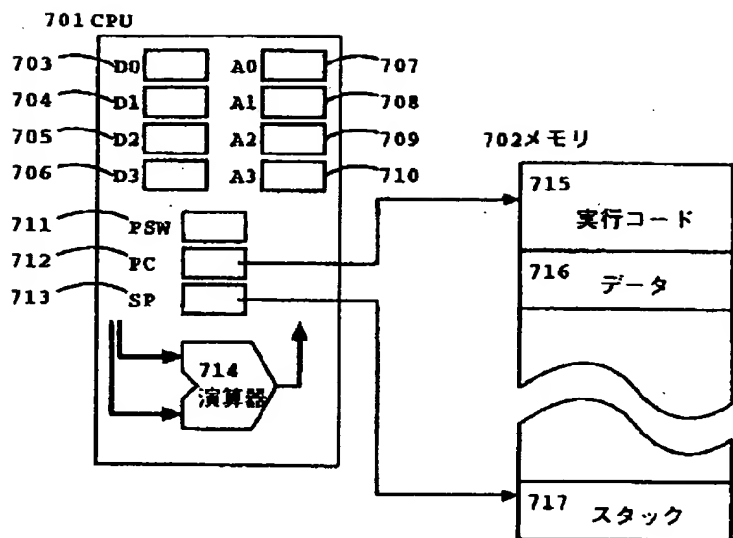
【図3】



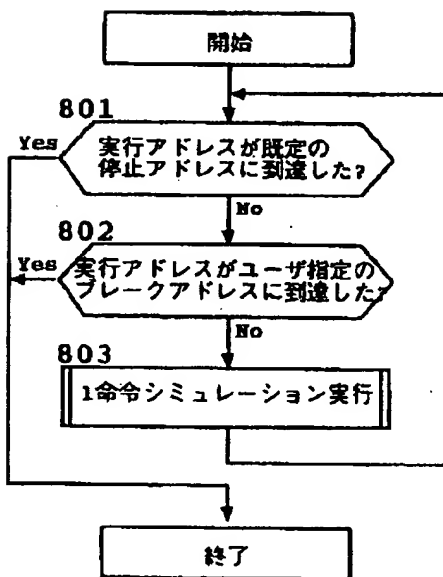
【図2】



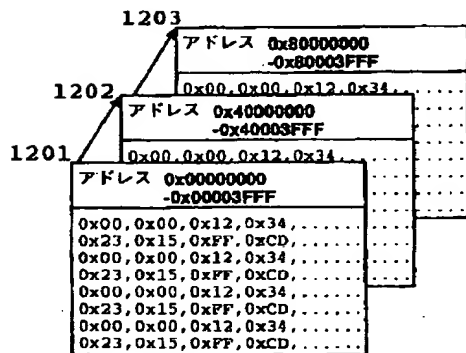
【図4】



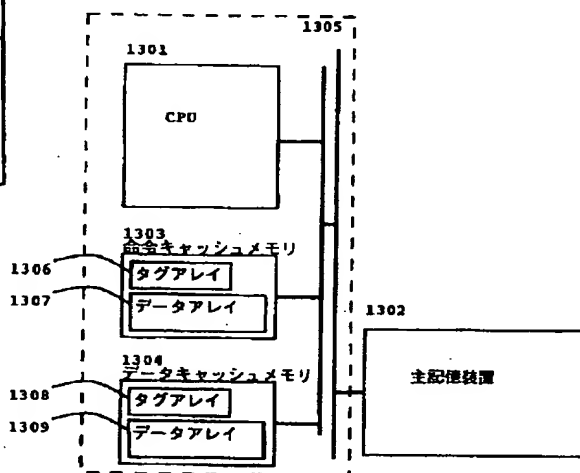
【図5】



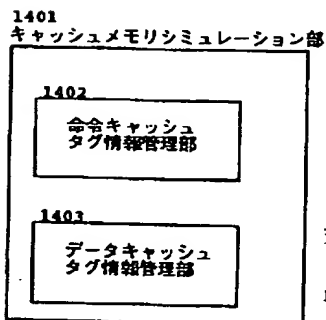
【図8】



【図9】



【図10】



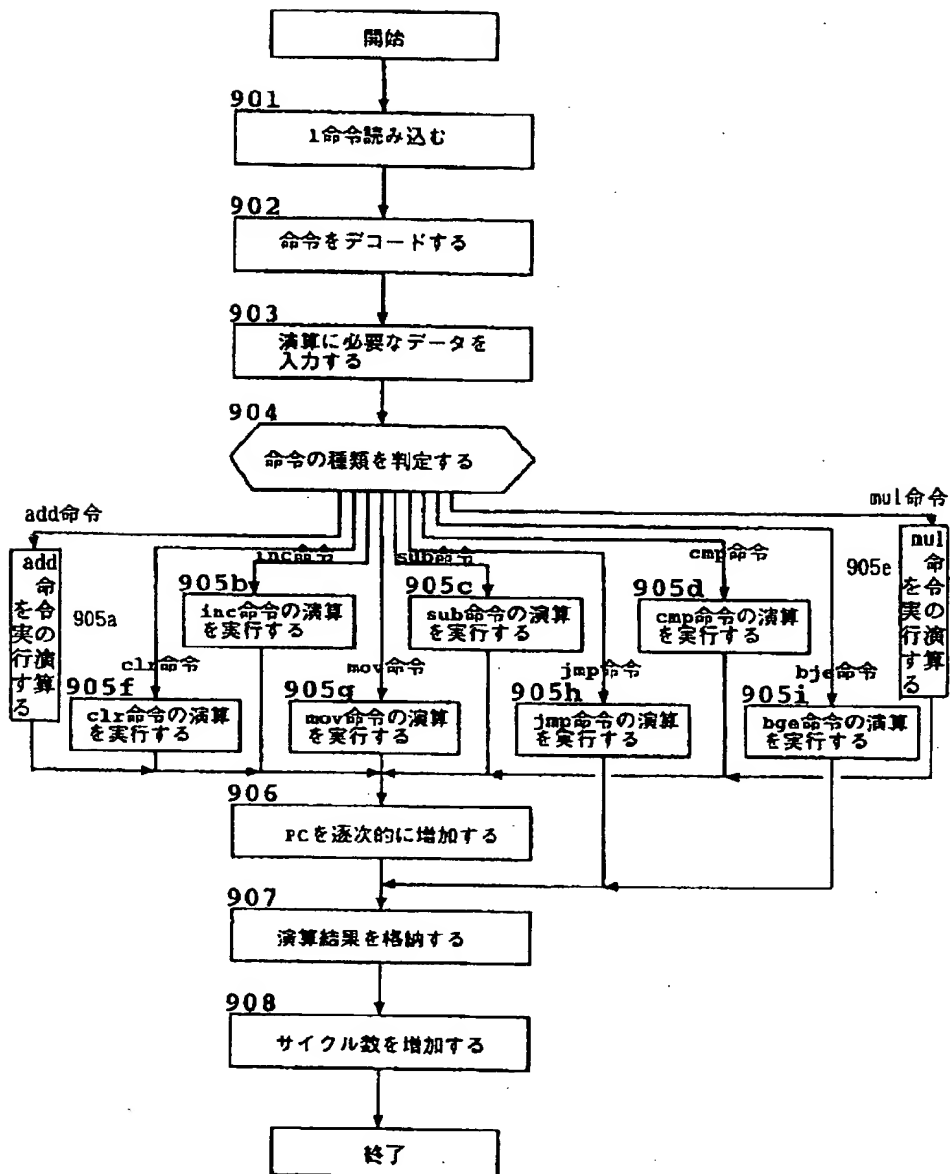
【図11】

エントリ	有効	タグ
0	○	0x40001000
1	○	0x40001010
2	—	
3	○	0x40002020
...
126	○	0x400007E0
127	○	0x400017F0

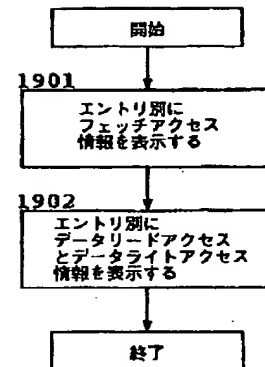
【図19】

エントリ	シンボル	ヒット		ミス	
		リード	ライト	リード	ライト
0x40	変数x	0	0	0	101
	変数1@main	300	100	100	1
0x41	変数y	100	100	0	1

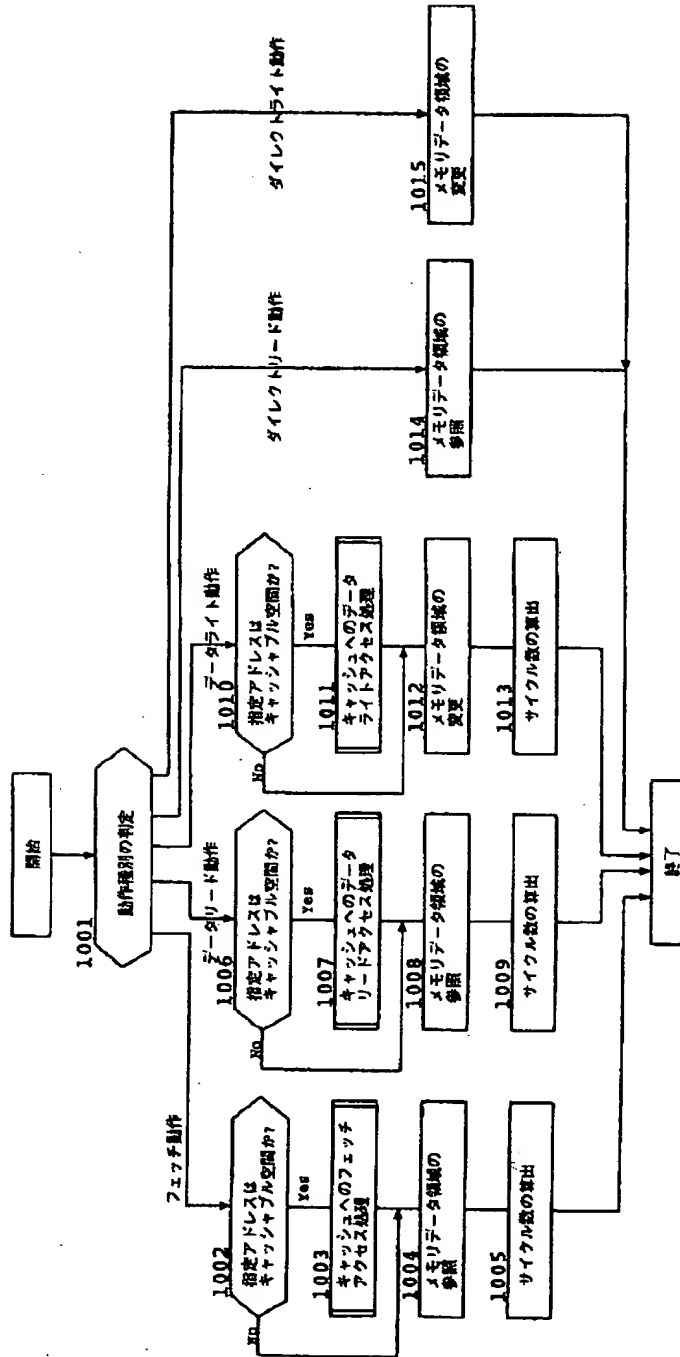
【図6】



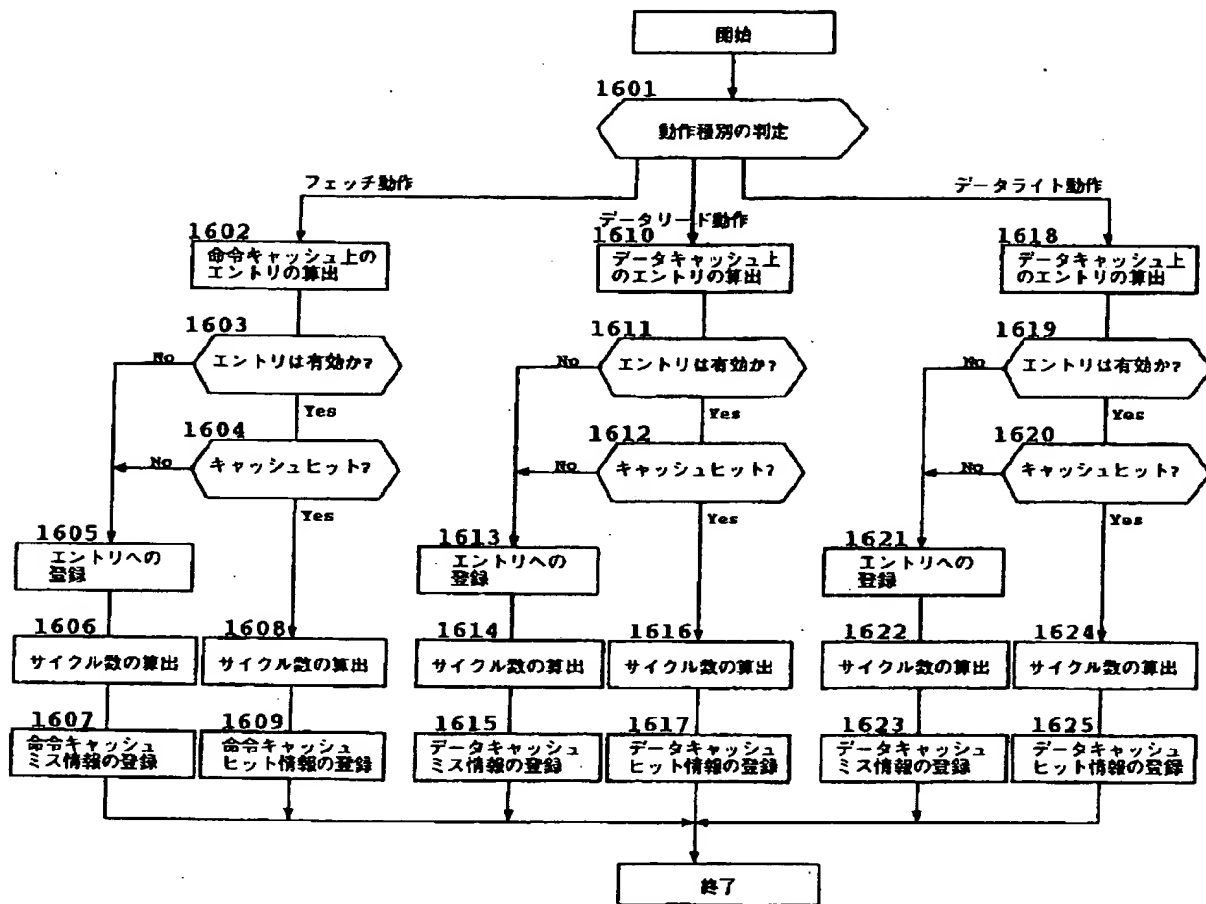
【図18】



【図 7】



【図 12】



【図 1 3】

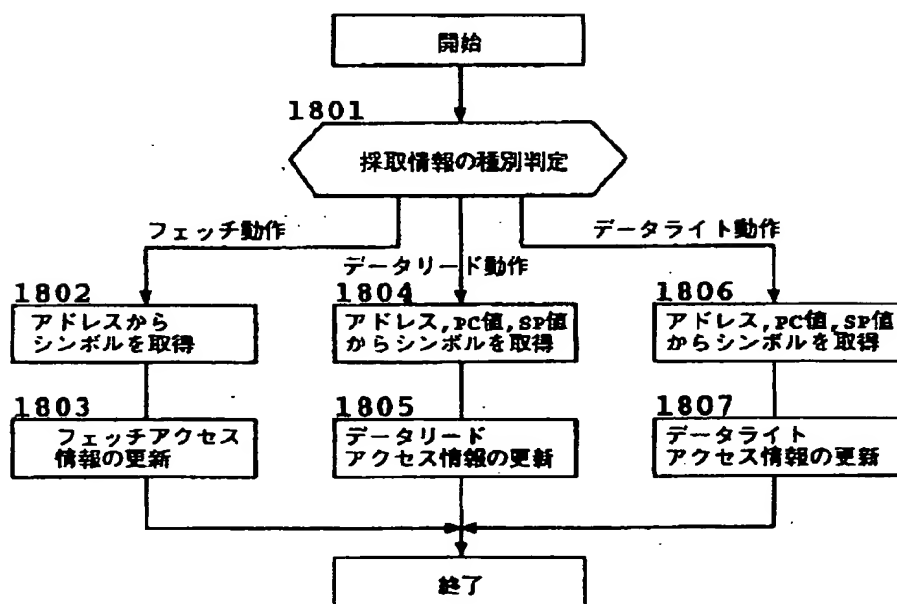
1701
キャッシュアクセス情報採取部

1702 命令キャッシュ アクセス情報管理部						
1704 フェッチアクセス情報						
entry	アドレス	Hit	データ長	シンボル	回数	割合
0x12	0x00000120	Miss	4	関数main()	1	0
0x12	0x00000124	Hit	4		1	-
0x12	0x00000128	Hit	4		1	-
0x12	0x0000012c	Hit	4		1	-

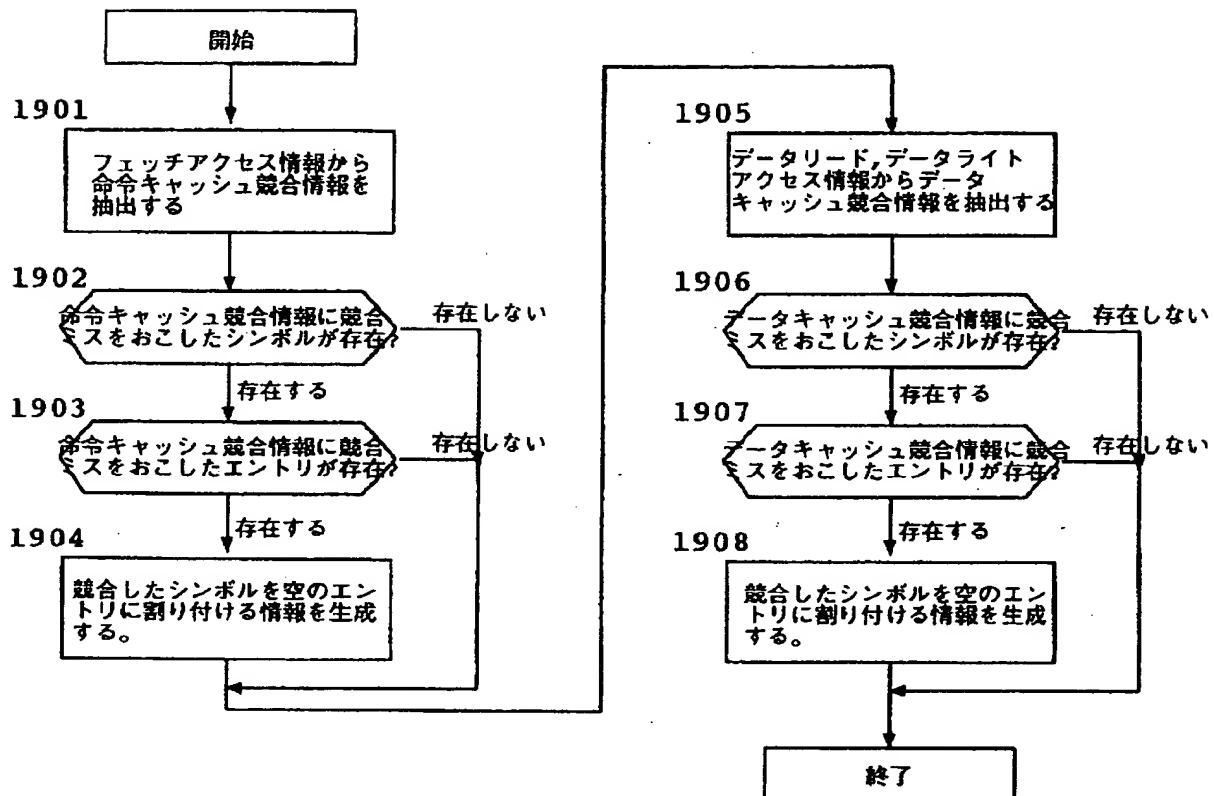
1703 データキャッシュ アクセス情報管理部						
1705 データリードアクセス情報						
entry	アドレス	Hit	データ長	シンボル	回数	割合
0x40	0x00003400	Miss	4	変数x@main	100	99
0x40	0x00003400	Hit	4	変数x@main	300	-
0x41	0x00000410	Hit	4	変数y	100	-

1706 データライトアクセス情報						
entry	アドレス	Hit	データ長	シンボル	回数	割合
0x40	0x00000400	Miss	4	変数x	101	101
0x40	0x00003400	Hit	4	変数x@main	100	-
0x40	0x00003400	Miss	4	変数x@main	1	1
0x41	0x00000410	Miss	4	変数y	1	1
0x41	0x00000410	Hit	4	変数y	100	-

【図 1 4】



【図15】



【図16】

(a)

2101 命令キャッシュ競合情報管理表

1706

entry	シンボル	競合ミスをおこしたシンボル
0x00		
0x01		
⋮		
0x10		
0x11		
0x12	関数main()	
⋮		
0x7E		
0x7F		

(b)

2102 データキャッシュ競合情報管理表

entry	シンボル	競合ミスをおこしたシンボル
0x00		
0x01		
⋮		
0x40		変数x, 変数1@main
0x41	変数y	
0x42		
⋮		
0x7E		
0x7F		

【図 1 7】

